**Determining the Necessity of Max-Cardinality Ordering During Tree Decompositions**

CSCE 421

*Tomo Bessho*

April 29th, 2020

**Overview**

For my final project, I chose to create a tree decomposition of the CSP and to determine whether it was necessary to run the max-cardinality algorithm before running the max-clique algorithm. From class, we learned that the trees decomposed from this process can be used to efficiently find the solutions to the CSP in a backtrack-free manner. To create these decompositions, I first need to manifest the CSP into a graph with vertices and edges. To do this, each variable is represented as a vertex in the graph and every binary constraint is represented as an edge. The unary constraints are excluded from this graph, as they would not create an edge in the graph. For every edge, the two vertices of the edge are the two variables in the corresponding scope of the constraint.

Once this was accomplished, the next task was to implement the min-fill heuristic to triangulate the graph. Not only does the min-fill algorithm put the variables in the perfect elimination ordering (PEO), but it also adds in edges to the graph to make sure that the graph is triangulated in a manner that minimizes the number of edges filled in.

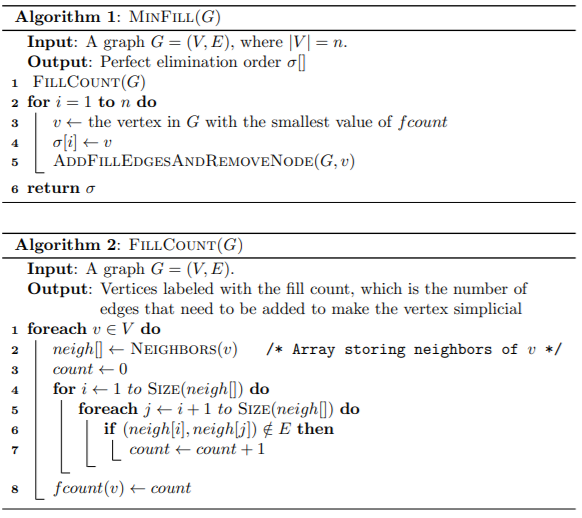
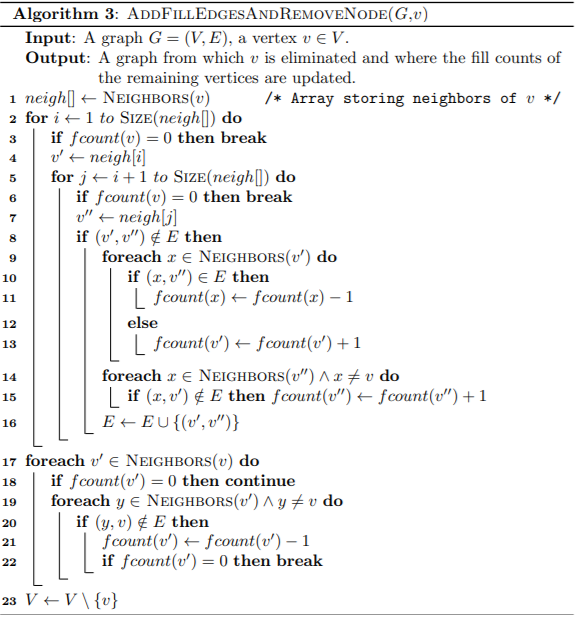
With the graph triangulated, the next task was to implement the max-cardinality ordering. This algorithm also returns the variables in a PEO. The reason I implemented max-cardinality was to determine whether a triangulated graph already in PEO needed to be put through the max-cardinality algorithm to return a joining tree.

With the two algorithms for getting a PEO established, the next task was to implement the max-clique algorithm. This algorithm takes the variable in the PEO and returns the clusters with the largest number of variables in each cluster.

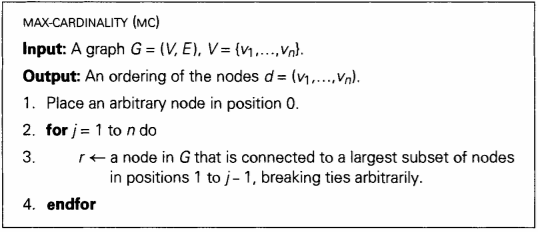
Lastly, I then implemented the joining tree algorithm to get the desired tree decomposition structure of the CSP. The joining tree takes all the max-cliques from before, finds the most optimal tree structure so that all the clusters are connected.

**Pseudocodes**

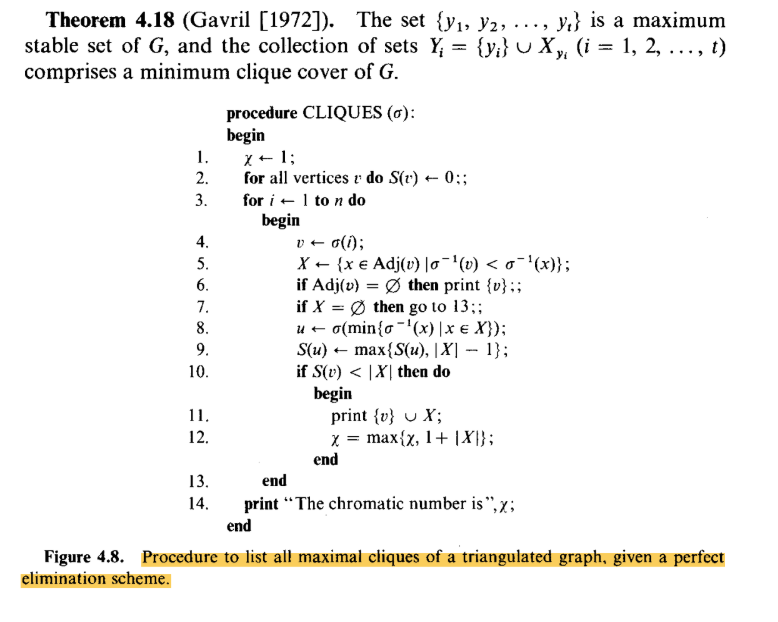
In this section, I have outlined all the pseudocodes I have used in this assignment. The minfill algorithm has a total of 3 algorithms, where Algorithm 1 is defined as the main function and Algorithm 2 and 3 are used as helper functions for Algorithm 1. Similarly, the Component Classifier Algorithm uses a helper function defined as DFS(v) that the main function utilizes to run DFS (depth-first search) from a vertex.

**Minfill Algorithm [Kjærulff, 1990]**[](https://github.com/tzxb018/csp-solver/blob/master/minfill1.png) [](https://github.com/tzxb018/csp-solver/blob/master/minfill2.png)

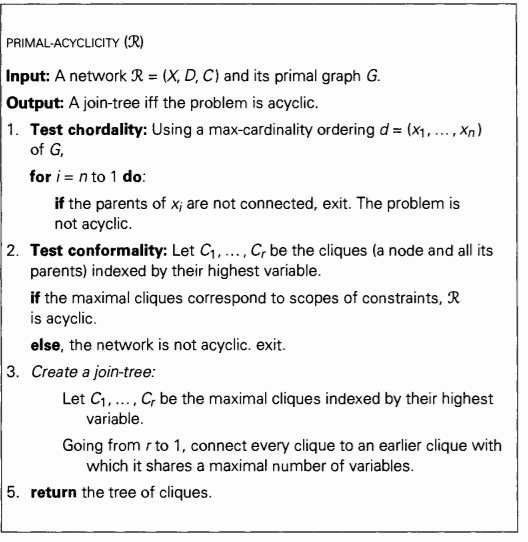
**Max-Cardinality Algorithm [Tarjan and Yannakakis, 1984]**

[](https://github.com/tzxb018/csp-solver/blob/master/maxcard.png)

**Max-Clique Algorithm [Golumbic, 1980]**

[](https://github.com/tzxb018/csp-solver/blob/master/maxclique.png)

**Join Tree Algorithm [Dechter, 2003a]**

[](https://github.com/tzxb018/csp-solver/blob/master/jointree.png)

**Component Classifier Algorithm [Hopcroft, 1973]**

A screenshot of a cell phone

Description automatically generated

**Compiling**

The main method to use to compile this project is still the same. The MyParser.java file should be used to compile the projects. In the MyParser.java file, the file will look for all the flags given and the arguments behind the flags, which should be separated by spaces. The flag needs to be placed before the argument itself; however, the ordering of which flags does not matter. Here is an example: -f ~/xmls/zebra-supports2.xml -s BT -u LX. The following are the flags built into the MyParser class:

* -f: path to the XCSP file
* -s: type of search algorithm (currently, the options include: BT (backtracking), CBJ (conflicted-backtracking), FC (forward checking), FCCBJ (hybrid between FC and CBJ))
* -u: type of variable sorting heuristic (options include: LX (lexicographical ordering), LD (least domains), DEG (degree ordering), and DD (domain to degree ratio ordering), W (min-width ordering), MF (min-fill ordering), MC (max-cardinality ordering))
* -a: type of arc consistency algorithm (options include: ac1, ac3)
* -t: type of tree decomposition (options include: TD (for tree decomposition without Max-Cardinality), and TD-MC (for tree decomposition with Max-Cardinality))

To run the tree decomposition, only the file name flag and the method of tree decomposition needs to be inputted in the arguments. For example, the following is how one can run tree decomposition: -f <path/to/file.xml> -t TD.

**Experimental Method**

To determine whether max-cardinality was necessary, I ran two simulations, one where the ordering from min-fill is used as the ordering for finding the max-cliques, which is then used in the joining tree algorithm. The second method takes the ordering from max-cardinality after running min-fill and using that ordering as a parameter for the max-clique algorithm to build the joining tree. These two methods were then tested on both the CSPs we have been using for class assignments and some benchmark problems from the 2008 CSP Competition (found at <https://cse.unl.edu/~consystlab/resources/CPAI08benchmarkstats.html>).

**Problems Encountered**

Before starting this project, I had not properly assigned the neighbors of all the variables. Instead of adding the edge for each constraint linearly, I was going through each variable, checking which constraints had the variable in the scope to determine the edges of the graph. This created a lot of value/reference errors and did not find all the edges. Whenever I needed to find a neighbor of a neighbor of a vertex, it would return empty. To fix this, I went through each constraint and added the two variables in the scope as neighbors of one another. A simple fix integral to the continuation of this project.

When first implementing min-fill, when taking the vertex out of the graph, I was not properly taking out the edges incident to the removed vertex. With this, whenever the min-fill algorithm searched for the neighbors of the certain edge, if the edges incident to the vertex removed were not also taken out, it would detect the removed vertex as a neighbor, causing an error in the algorithm.

The last major error I encountered was not considering disconnected graphs. Both the algorithms and I assumed that the graph being passed in was connected; however, this will result in some wonky and inaccurate results when trying to build the tree. To fix this, I implemented a simple DFS algorithm that would 'visit' all the nodes in a component and run the tree decomposition algorithms for each component. In the case where the graph was initially connected, this step does not affect the results.

**Evaluation and Conclusion**

The number of edges filled in by min-fill, the number of maximal cliques returned from the max-clique algorithm, the treewidth (size of the largest clique - 1), and the largest number of variables in the separators of the joining tree. The basic CSPs from our class assignments as well as some benchmark problems were tested and their data from this evaluation method was then put into an excel sheet (found in /csp-solver/Output Excel Files/Bessho-TreeDecomp-Results-Updated.xlsx). In addition, the number of variables, edges, and the density of the problem was reported as well.

When running the two different methods, the results of all 4 of these categories of data showed that there is *no difference* between running minfill and going directly to max-clique and running both min-fill and max-cardinality before going to max-clique. To see this, I created a column called ‘Match?’ (Column P of Sheet 'Benchmark Problems') that returns true if the values in each column for running tree decomposition without max-cardinality match every value in the columns for running tree decomposition with max-cardinality. From all the problems I ran, this column has been calculated to be true, meaning their results are all the same regardless of running max-cardinality or not.

The error column (column T) compares my results with those in the 2008 site's results. There are some cases where the separator or the treewidth is off by 1 or 2 (please look at notes below regarding these minimal errors).

**Important Notes Regarding Results**

The results I obtained from the Benchmark had incorrectly calculated the treewidth. Instead of calculating the treewidth by finding the size of the largest clique and subtracting 1, they simply reported the treewidth to be the size of the largest clique. Also, some of the results from the 2008 site doesn't match the Stampede database (can be found at <http://consystlab.unl.edu/benchmarks/>); however, in the cases where my results do not match the 2008 site, my results match with the Stampede's database exactly.

**Future Work**

Although this program successfully decomposes the CSP into a tree, it still doesn't actually solve the CSP. Thus, using the tree decomposition to solve the CSPs is another task that could be done. As stated before, the tree decomposition algorithms currently only find and output the results of the tree decomposition and do not use these structures to solve the CSP. By solving a subproblem of the CSP in each cluster and taking advantage of the tree structure to prevent backtracking during search, the CSP could be solved much more efficiently than the previous search algorithms, such as BT, CBJ, FC, etc.

**Citations**

Hopcroft, J.; Tarjan, R. (1973), "Algorithm 447: efficient algorithms for graph manipulation", Communications of the ACM, 16 (6): 372–378.

Martin C. Golumbic. Algorithmic Graph Theory and Perfect Graphs. Academic Press Inc., New York, NY, 1980.

Rina Dechter. Constraint Processing. Morgan Kaufmann, 2003.

Robert Endre Tarjan and Mihalis Yannakakis. Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hyper- graphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

U. Kjærulff. Triagulation of Graphs - Algorithms Giving Small Total State Space. Research Report R-90-09, Aalborg University, Denmark, 1990.